DA－90－13

論理関数の種々の分解手法を統合した LUT 回路合成法

山下 茂　澤田 宏　名古屋 彰

NTT コミュニケーション科学研究所
〒 619-0237 京都府相楽郡精華町光台 2-4
Tel: 0774-93-5275　Fax: 0774-93-5285
E-mail: {ger, sawada, nagoya}@cslab.kecl.ntt.co.jp

あらまし　本稿では，LUT(look-up table) の回路を合成する一手法を紹介する．紹介する手法は既存の手法とは異なり，代数的な分解手法や関数分解手法などの様々な分解手法を全て利用して回路を合成する．回路合成の途中で種々の分解の中から最良であろう分解を選択するのに，我々は過去に設計された回路の情報を記述したコスト定義ファイルと呼ばれるファイルを利用する．

提案手法は，種々の分解手法を利用した LUT 回路の合成手法のフレームワークと考えられる．また，実験結果は極めて良好である．

キーワード　関数分解, 代数的分解，FPGA, ルックアップテーブル

A Method for Synthesizing LUT Networks Integrating Various Decomposition Methods

Shigeru Yamashita,　Hiroshi Sawada,　Akira Nagoya

NTT Communication Science Laboratories
2-4, Hikaridai, Soraku-gun, Seika-cho, Kyoto, 619-0237 Japan
Tel: +81-774-93-5275　Fax: +81-774-93-5285
E-mail: {ger, sawada, nagoya}@cslab.kecl.ntt.co.jp

Abstract　This paper presents a method for synthesizing look-up table (LUT) networks. The strategy employed by our method is very different from the strategies of previous methods; many decomposition methods that are not only algebraic but also functional are integrated very well. To select a possibly best decomposition form for an intermediate decomposition, we use a **cost data base file** which is generated from the previous decomposed networks.

Our method can be thought of as a general framework for LUT network synthesis integrating various decomposition methods. The experimental results are very encouraging.

key words　functional decomposition, algebraic decomposition, field programmable gate array (FPGA), look-up table (LUT)

1 はじめに

組合せ回路をルックアップテーブル（LUT）ベースのFPGA（Field Programmable Gate Array）にマッピングしようとする場合，最初に所望の論理を実現する各ノードがK（通常4から6）以下の入力数の論理関数を実現する回路[1]を作らなければならない．現在までに提案されているK-LUT回路の合成方法は，以下の二つのカテゴリに大別することができる．

第一のカテゴリの手法は，通常のスタンダード・セル向けの論理合成手法をそのまま適用したものであり，以下のような手順を経る．

- 通常のスタンダード・セル向けの論理合成手法を用いて，各ノードがK入力以下となるように分解を行う．多くの手法は，論理表現を代数式として扱って式を変形することによって分解を行う．本稿では，そのような手法を**代数的**分解手法と呼ぶことにする．この手法として有名なものに，kernel extraction[1]等がある．
- 各ノードの入力数がKを越えないという条件下で，テクノロジマッパを用いてノードを出来るだけマージしてノード数を減らす．

このカテゴリの手法では，Chortle-d[2]，MIS-pga-delay[3][2]およびFlowMap[4]等が有名である．ノードのマージに関しては，特定の条件下での最適なアルゴリズム[2, 4]が提案されている．このカテゴリの手法は既存の論理合成手法から容易に拡張できるためよく用いられるが，後で述べる第二のカテゴリの手法の方が品質の良いK-LUT回路を合成出来る場合が多い．その理由として以下の様なものが考えられる．

- テクノロジ独立の段階での論理設計は，積和形で表現した論理式に現れる文字の数をコストとして扱うことが多い．CMOSテクノロジの場合，このコストは妥当だと考えられるが，LUTの場合必ずしもそうではない場合がある．
- テクノロジ独立の段階で設計された回路が最終結果にとって支配的であり，テクノロジマッパの最適化が有効に働かない場合がある．

第二のカテゴリの手法は，回路の出力関数をOBDD（ordered binary decision diagram）等で表現して，直接分解することを基本とする[5, 6]．本稿では，関数を直接分解する手法を**関数分解手法**と呼ぶことにする．このカテゴリで用いられる関数分解は，Roth-Karp分解[7]と呼ばれる特定の分解の形を基本とする場合が多いが，別の分解の形を基本とする関数分解を用いる方が最終的な結果が良くなる場合もあるという報告もある[8]．

我々も種々の関数分解手法を開発してきた[5, 8, 9]．本稿では，これらをうまく組合せることによって，単独で用いるよりもより良いK-LUT回路を合成する手法について述べる．本稿で提案するK-LUT回路合成法は，多くの分解手法を取り入れてその中から最適と考えられる手法を選択することを基本とする．

提案手法は，種々の分解手法を効率的に組み合せてK-LUT回路を合成するフレームワークの提案と考えられる．実験結果によりこのアプローチが有効であることを示す．

2 LUT回路合成手法

2.1 問題の定式化

我々の目標は，与えられた組合せ論理を実現するK-LUT回路の中で，以下のように定義されるコストの最小な回路を合成することである．

コスト＝回路内のノード数＋W×回路の段数

（Wはユーザー定義の重み定数）

2.2 我々の手法の方針

論理合成の分野では，大きな入力数のノードをより小さな入力数のノードに分解することは重要な操作の一つである．そのため，数多くの分解手法が提案されてきた．例えば，disjoint decomposition[7]，non-disjoint bi-decomposition[8], kernelを用いたweak division, davio展開等がある．これらの手法のうちのいく

[1]以下では単にK-LUT回路と呼ぶ．

[2]MIS-pgaは部分的に，代数的分解手法以外の手法も用いている．

つかは既存のK-LUT回路の合成手法にも使われている．本稿で提案するK-LUT回路の合成手法は，このような種々の分解手法が利用できることを前提として，以下の考えに基づいている．

「各ノードがK入力以下になるまで，分解手法を繰り返し適用してK-LUT回路を合成する．用意された分解手法の考えられる全ての組合せの分解を行って合成された全ての回路を列挙する．列挙された全ての回路において後処理として，K入力以下のノードのマージを行う．以上の操作で合成される全ての回路の中に最小コストのK-LUT回路が存在するはずである．」

しかし，全ての分解手法の全ての組合せの可能性を調べることは非現実的である．そのため我々の手法では一つのノードを分解する際に，最良であろう分解をその都度採用していくことにする．通常のK-LUT回路合成手法では，全ての分解の後でK入力以下のノードをマージしてノード数を減らすという処理を行う．しかし，もし分解を全て行った後でノードのマージを行う方針をとれば，1章で述べたようにそれぞれが最適な手法であったとしても最終的に最適なK-LUTの回路が出来るかどうかわからないというジレンマに陥る．そこで我々の方針では，ある分解において最良の分解を選ぶ際にノードのマージも同時に行うことにする．

また，分解をする際に複数ノードの共有は考慮に入れない．そのような分解も考慮に入れると評価しなければならない分解が多くなり過ぎるからであり，さらに中間的な論理の共有を行わない方が回路の段数は小さくなると考えられるからである．ただし，我々の提案手法においても，追加的処理として中間的な論理の共有を行うことは可能である．これについては3章で述べる．

我々の提案手法の方針の特徴は以下のようにまとめることが出来る．

- 最良と思われる分解手法を選択する際には，ノードのマージの効果も考慮する．
- 中間的に論理を共有するような分解手法は，考慮に入れない．

この方針により，我々は多くの分解手法を統一的に扱うことができる．

2.3 合成手法の概要

提案する合成手法の概要は以下の通りである．

ステップ1 与えられた組合せ論理回路の各出力に対して一つのノードを生成する．各ノードは回路の入力変数を直接の入力に持つノードとなる．各ノードに対して，与えられた回路の表現から以下のデータを用意する．

- OBDDで表したノードの出力関数．(OBDDのノード数が制限値を越えるような場合は生成しない．)
- 多段又は2段の積和論理で表したノードの出力関数．
- OBDDで表したノードの出力のドントケア．

OBDDで表現した関数と積和論理の式の両方を用意することにより関数分解手法と代数的分解手法の両者が適用可能となる．

ステップ2 入力数がKを越えるノードが存在すれば，そのノードに対して以下を行う．回路内にそのようなノードが存在しなくなれば，ステップ3へ．

ステップ2.1 種々の分解手法の中で最適と考えられる手法で該当ノードを分解する．

ステップ2.2 該当ノードの分解によって新たに生成されたノードの入力数がKを越える場合更にそのノードの分解が必要なため，ステップ1と同様のデータをそのノードのために用意する．分解によっては，分解後のノードの論理式表現が得られない場合がある．そのような場合には，論理関数から文献[10]の手法により非冗長な積和論理表現を生成しておく．

ステップ3 生成された回路から，2.6章で述べる**コスト定義ファイル**を新たに作る．

計算時間または必要なメモリ量の観点から，入力数の多い関数には適用困難だと考えられる分解手法も存在する．そのため，分解すべきノードの入力数がある閾値より多い場合には，ステップ2.1においてそのような手法は調べないようにする．

2.4 種々の分解手法の統一的な取り扱い手法

まず，種々の分解手法の中からいくつかの手法について簡単にまとめる．

disjoint decomposition

与えられた論理関数fに対して，X^BおよびX^Fを互いに素な変数集合とした時，$f = \alpha(g_1(X^B), \ldots, g_t(X^B), X^F) = \alpha(\vec{g}(X^B), X^F)$という分解をdisjoint decompositionという[7]．この分解は，fをOBDDで表現すれば比較的容易に検出することが可能である[5, 11]．

non-disjoint bi-decomposition

X^1およびX^2を互いに素な変数集合と制限せずに，$f = \alpha(g_1(X^1), g_2(X^2))$という分解を効率良く見つける手法が文献[8]で提案されている．この手法は，fが不完全定義論理関数でも直接扱うことができ，分解結果のg_1およびg_2も不完全定義論理関数となる．

kernelを用いたweak division

与えられた論理式Fを，論理式PとQに共通変数がないという制限で，$F = Q \cdot P + R$という形の代数的な割り算により分解することをweak divisionという．特に，除数Pにカーネルと呼ばれる特殊な論理和形を用いる手法が有効である[1]．

Davio展開およびShannon展開

与えられた関数fおよびある変数x_iについて，$f_0 = f|_{x_i=0}$，$f_1 = f|_{x_i=1}$，$f_2 = f_0 \oplus f_1$とした時に，$f = f_0 \oplus x_i \cdot f_2$，$f = \overline{x_i} \cdot f_2 \oplus f_1$，$f = \overline{x_i} \cdot f_0 + x_i \cdot f_1$という分解をそれぞ正極性Davio展開，負極性Davio展開，Shannon展開という．これらの展開は，必ず入力数の少ない関数へ分解可能であるという点で重要である．

我々の手法では，ノードn_iの分解を「ノードn_i'とその直接の入力となる複数のノード$n_{i1}, \cdots n_{in}$を新たに導入して，n_iの出力をn_i'の出力に置き換える」という形で実現する分解手法を扱う．この分解形式の例として，図1(b)にnon-disjoint bi-decompositionの例を示す．この形式に当てはまらない一部の分解手法も存在するが，上述した種々の分解手法を含め多くの分解手法はこの形式に当てはまるため我々の手法に容易に組み込み可能である．また，以下ではある分解において新しく導入されたノードの集合を**DecompArea**と呼ぶことにする．例では，DecompAreaは図1(b)の点線で囲まれた領域のノードの集合である．

2.5 最良の分解手法の選択方法

2.3で述べた手法のステップ2.1において，我々はあるノードn_iのその時点で最良と考えられる分解を選択する．具体的には，以下で述べる方法で分解の**コスト**を計算し，コストが最小となる分解を選択する．

2.5.1 コスト計算の前処理

我々の方針は，「種々の分解手法を用いるが全ての分解の後にノードのマージは行わない」というものである．そのため，ある分解において新たに導入されたノードがK入力未満であり最終的にそのノードを他のノードとマージしてもK入力以下のままであるような場合も考慮に入れてコストを計算しなければならない．そこで，分解のコストを計算する前に以下の処理を行う．分解後のDecompAreaの中に，K入力未満でかつそのノードの入力または出力がつながっているノードにそのノードをマージしても入力数がKを越えないようなノードが存在すれば，そのノードを他のノードとマージしてDecompAreaより取り除く．図1の例では，図の(c)で示すようにn_i'とn_{i2}はDecompAreaに含まれないノードとマージ可能で，これらのノードをDecompAreaから除いてコストの計算には含めない．

2.5.2 分解のコスト計算

上述したマージの処理を行った後のDecompArea（図1(c)の点線で囲まれた領域に相当）に含まれるノードに対して，分解のコストは式(1)で計算される．

ここで，$LEV(n_j)$は式(2)により再帰的に計算されるノードn_jの出力を実現するために最終的

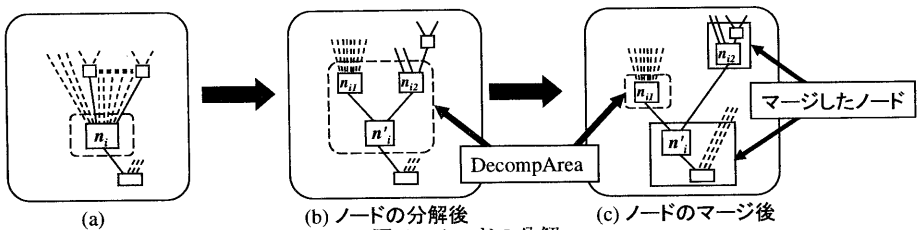

図 1: ノードの分解

$$\text{分解のコスト} = \{\sum_{n_j \in DecompArea} CostLUT(n_j)\} + W \times \{\max_{n_j \in DecompArea} LEV(n_j)\} \quad (1)$$

(W はユーザー 定義の重み定数)

に必要となる段数の予想値である．式 (1)，(2) に現れる $CostLUT(n_j)$ および $CostLEV(n_j)$ は，ノード n_j だけを K-LUT を用いて実現した時に必要となる段数と LUT 数の予想値である．n_j の入力数が K 以下ならこれらの値は 1 である．しかし，その他の場合には実際に設計してみなければ厳密にこれらの値を計算することは出来ない．そこで我々の手法ではこれらの値は，コスト定義ファイルと呼ばれる過去の設計データの値を保存しているデータベースを参考にして決定することにする．コスト定義ファイルについては次章で述べる．

2.6 コスト定義ファイル

我々の手法ではノード n_i が与えられた時，$CostLUT(n_i)$ および $CostLEV(n_i)$ を以下のように予想する．

ステップ 1 n_i の内部論理の関数の複雑さを特徴づけると考えられるパラメータの値を計算する．例えば，関数の入力数，関数の論理式表現での積項数や文字数等を用いる．

ステップ 2 後述するコスト定義ファイルのデータベースに存在するエントリーの中でステップ 1 で計算したパラメータの値に最も近いエントリから，$CostLUT(n_i)$ および $CostLEV(n_i)$ を決定する．

コスト定義ファイルは以下のようにして作られる．

- 設計データが存在しない段階では，$CostLUT(n_i)$ と $CostLEV(n_i)$ に以下の値を持つコスト定義ファイルを生成する [6].

$$\begin{cases} 1 & (n_i\text{の入力数が}K\text{以下}) \\ (n_i\text{の入力数}) - K + 1 & (\text{その他の場合}) \end{cases}$$

- 上述したコスト定義ファイルを用いて種々の回路を設計する．設計した回路の各ノード n_i に対して，回路の入力に対する n_i の出力関数を特徴づけるパラメータの値を計算する．それらのパラメータの値と，n_i の入力側からのレベルおよび n_i の入力側のノードの総数を $CostLEV(n_i)$ と $CostLUT(n_i)$ に関連づけたエントリーを新しいコスト定義ファイルに追加する．

以上をまとめると，コスト定義ファイルには論理関数を特徴づけるパラメータの値とその論理関数を実現するのに実際に必要となった LUT の総数と段数の統計的な関係が記述されているということである．そして，我々の手法ではファイルに記述された統計値から最も近そうな値を $CostLEV(n_i)$ と $CostLUT(n_i)$ に採用するということである．

$$LEV(n_j) = \begin{cases} 0 & n_jが回路の入力の場合 \\ \{\max_{n_k \in n_jの入力} LEV(n_k)\} + CostLEV(n_j) & その他の場合 \end{cases} \quad (2)$$

3 提案手法からの発展

2章で述べた提案手法は，以下のように発展させることが可能である．

3.1 分解手法の前処理

その分解が存在するならば適用する方が良く，また比較的容易に検出可能な分解の形が存在する[12]．我々の手法のステップ2.1に先だって，そのような分解の形を見つける手法のみを必ず適用するようにすれば，全体の処理時間の短縮が期待できる．

3.2 内部論理の共有

我々の手法では，回路内部での論理の共有についてほとんど考慮されていない．それは，種々の分解手法を統一的に組み込むためと論理の共有を行うと段数が増加することが多いと考えられるからである．しかし，実際には回路内部で論理の共有を行えば回路規模が小さくなる場合も多い．そこで，回路内部で論理の共有を行うために我々の手法に以下の二通りの手法を追加することが考えられる．

- ノードn_iの分解を行う際に，既に存在するノードn_jをn_iの入力に加えることによってn_iの入力数を減らすことが出来ることがある．そのような手法を一つの分解手法と考えて我々の手法に組み込むことが可能である．ただしそのような分解では，分解のコストの計算の際にn_jはDecompAreaには含めない．

- K-LUTの回路において，あるノードの出力を別のノードの出力と置き換えても回路全体の論理が変更しないかを調べる手法がある[13]．その手法を後処理として用いることで，我々の手法で生成したK-LUT回路内部のいくつかのノードの共有を行うことが可能である．

3.3 分解手法の評価の並列化

我々の手法は該当ノードに対して種々の分解手法を全て試みて評価するという方針をとるため，多くの計算時間を必要とすると思われる．しかし，ステップ2.1における種々の手法の分解の評価はそれぞれを別々のプロセッサに割当てて並列処理させることが容易だと考えられる．また，あるノードの分解の際にある分解手法だけ他と比べて異常に長い時間を必要とする場合，その分解結果もそれほど良くないことが多いと思われる．そのため，並列実行を行う場合には，他と比べて異常に長く時間を要している分解手法については適用候補からはずして途中で中断をすることにより，全体の処理時間をより短縮することが可能であると考えられる．

4 実験結果

4.1 LUT回路合成の結果比較

提案手法で5-LUT回路を合成した結果を表1に示す．また比較のため，LUT回路合成手法の中で少ない段数の回路を合成すると言われている有名な合成手法の結果も表に示す．表中の“♯lut”および“♯lvl”は，それぞれ合成結果の回路内の5-LUTの数および回路の段数を示す．“CPU”は，Sun Ultra 2 2200上で実行した際の実行時間（秒）を示す．我々の手法の結果と他の手法の結果を直接比較するために，同じ回路に対しての合計値を表の下部に示した．

比較結果より，我々の手法は最もロバストであることがわかる．種々の分解手法を組合せているので，どのような場合でもそれほど悪い回路を合成しないことがわかる．

実験ではそれほど顕著な例が見られないが，場

表 1: 5-LUT 回路合成の結果比較

circuit	ALTO[14]		mispga-d		chortle-d		FlowMap-r		BoolMap-D[6]		提案手法		
name	#lut	#lvl	#lut	#lvl	#lut	#lvl	#lut	#lvl	#lut	#lvl	#lut	#lvl	CPU
5xp1	19	2	21	2	26	3	23	3	13	2	11	2	0.33
9sym	7	3	7	3	63	5	61	5	7	3	5	4	0.55
alu2	61	6	122	6	227	9	148	8	43	4	33	4	5.44
alu4	259	8	155	11	500	10	245	10	268	7	85	7	77.33
apex4	-	-	-	-	1112	6	-	-	-	-	302	4	32.67
apex6	229	4	274	5	308	4	232	4	189	4	161	4	691.89
apex7	77	4	95	4	108	4	80	4	78	3	61	4	204.98
clip	33	3	54	4	-	-	-	-	-	-	11	3	2.39
count	47	3	81	4	91	4	73	4	42	2	30	4	732.5
duke2	156	4	164	6	241	4	187	4	193	5	150	4	162.75
f51m	15	3	23	4	-	-	-	-	-	-	10	3	0.34
misex1	14	2	17	2	19	2	15	2	15	2	10	2	0.22
misex3	251	6	-	-	-	-	-	-	-	-	166	6	196.64
rd73	8	2	8	2	-	-	-	-	-	-	6	2	0.21
rd84	13	3	13	3	61	4	43	4	10	2	7	3	0.54
sao2	38	3	45	5	-	-	-	-	-	-	21	3	3.56
vg2	26	3	39	4	55	4	38	4	30	4	21	4	120.16
z4ml	5	2	10	2	25	3	13	3	5	2	5	2	0.13
Total	1258	61	1128	67	2836	62	1158	55	893	40	1095	65	-
ALTO	1258	61									793	61	
mispga-d			1128	67							627	55	
chortle-d					2836	62					881	48	
FlowMap-r							1158	55			579	44	
BoolMap-D									893	40	579	44	

合によっては我々の手法は非常に多くの計算時間を要する事が予想される．しかし，3.3章で述べたように，並列処理によりその問題は容易に回避できると考えている．

どのような分解手法を用いるのか，およびどのようなコスト定義ファイルを用いるのかによって，我々の手法による結果は異なる．表1に示した結果は，以下の条件で合成した結果である．

- 関数分解としては，disjoint decompositions, non-disjoint bi-decomposition および davio 展開を用いた．
- まず，一旦2.6章で述べた最初のコスト定義ファイルを用いて表1に示した全ての回路を合成した．そしてその回路データ全てから生成した2番目のコスト定義ファイルを用いて合成した結果が表1である．

5 まとめ及び今後の課題

本稿では，種々の分解手法を組合せたLUT回路の合成法を述べた．提案手法は，代数的分解手法だけでなく関数分解手法も同じように組み入れることが可能であり，種々の分解手法を用いてLUT回路を合成するフレームワークと考えられる．提案手法は以下のような特長を持つ．

- 種々の分解手法を簡単に取り込むことが可能である．つまり，新しい分解手法を考案した場合その有効性をすぐに調べることが可能である．
- 用意する分解手法の組合せやコスト定義ファイルを変更することにより，同じ論理から容易に種々のK-LUT回路を生成可能である．
- 種々の分解手法を組合せるために，どのような場合でもそれほど悪い回路を生成することはない．

提案手法はその性質上多くの実行時間を必要とすることが考えられるが，並列実行の可能性等を考慮するとそれほど問題ではないと考えられる．

今後は，実設計データを提案手法により合成し，FPGAへのマッピング後の最終的な配置配線結果の評価を行いたい．

参考文献

[1] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Trans. CAD*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[2] R. J. Francis, J. Rose, and Z. Vranesic, "Technology Mapping of Lookup Table-Based FPGAs for Performance," in *Proc. ICCAD*, pp. 568–571, Nov. 1991.

[3] R. Murgai, N. Shenoy, and R. K. Brayton, "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," in *Proc. ICCAD*, pp. 572–575, Nov. 1991.

[4] J. Cong and Y. Ding, "An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," in *Proc. ICCAD*, pp. 48–53, Nov. 1992.

[5] H. Sawada, T. Suyama, and A. Nagoya, "Logic Synthesis for Look-up Table Based FPGAs Using Functional Decomposition and Support Minimization," in *Proc. ICCAD*, pp. 353–358, Nov. 1995.

[6] C. Legl, B. Wurth, and K. Eckl, "A Boolean Approach to Performance-Directed Technology Mapping for LUT-Based FPGA Designs," in *33rd ACM/IEEE Design Automation Conference*, pp. 730–733, June 1996.

[7] J. P. Roth and R. M. Karp, "Minimization Over Boolean Graphs," *IBM Journal*, pp. 227–238, Apr. 1962.

[8] S. Yamashita, H. Sawada, and A. Nagoya, "New Methods to Find Optimal Non-Disjoint Bi-Decompositions," in *ASP-DAC '98*, pp. 59–68, Feb. 1998.

[9] H. Sawada, S. Yamashita, and A. Nagoya, "Restricted Simple Disjunctive Decompositions Based on Grouping Symmetric Variables," in *Proc. of the seventh Great Lakes Symposium on VLSI*, Mar. 1997.

[10] S. Minato, "Fast generation of irredudant sum-of-products forms from binary decision diagrams," in *Proc. SASIMI*, pp. 64–73, Apr. 1992.

[11] Y.-T. Lai, M. Pedram, and S. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," in *30th ACM/IEEE Design Automation Conference*, pp. 642–647, June 1993.

[12] H. Sawada, S. Yamashita, and A. Nagoya, "Restructuring Logic Representations with Easily Detectable Simple Disjunctive Decompositions," in *Proc. of the Design, Automation and Test in Europe (DATE'98)*, pp. 755–759, Feb. 1998.

[13] S. Yamashita, H. Sawada, and A. Nagoya, "A New Method to Express Functional Permissibilities for LUT based FPGAs and Its Applications," in *Proc. ICCAD*, pp. 254–261, Nov. 1996.

[14] J.-D. Huang, J.-Y. Jou, and W.-Z. Shen, "An Iterative Area/Performance Trade-Off Algorithm for LUT-Based FPGA Technology Mapping," in *Proc. ICCAD*, pp. 13–17, Nov. 1996.